可持久化

2019年8月13日 21:43

一. 可持久化线段树/主席树

1. 每次插入时，不是从根1插，而是从新结点插
   i. 于是父子结点之间没有两倍序号关系了，只能存结构体指针或下标指向
   ii. 每次插入会更改一条logN的链，严格来说已经不是树了
   iii. 查询第[l,r]次插入之间的关系时，若插入的值满足可减性质，则可类似前缀和地用ask(r)-ask(l-1)，因为不同时期各节点的区间信息还是一样的，因此可以让ask函数多一个参数，一次调用计算两个端点
2. 值域区间内数据量树，用于离线查区间第k大数（洛谷P3834）（POJ2104）

```
int tot,rt[MN];     //每次更新的新树根
struct ST{
     int lc,rc;    //左右子结点编号
     int cnt;      //值域区间数据总数
}t[MN<<8];
//新建一个关于区间[l,r]的结点，返回其结点号
int build(int l, int r){
     int p= ++tot;//当前结点号
     ST &tp= t[p];//为了好写和好看，建一个引用
     if(l==r){
          tp.cnt= 0;   //初值为0，每次insert再+1
          return p;
     }
     int mid= l+r >>1;
     tp.lc= build(l, mid);
     tp.rc= build(mid+1, r);
     tp.cnt= t[tp.lc].cnt + t[tp.rc].cnt;
     return p;
}
//把关于[l,r]的原old结点的值[x]增加dx，返回新结点号
int insert(int old, int l, int r, int x, int dx){
     int p= ++tot;//当前结点号
     ST &tp= t[p];//为了好写和好看，建一个引用
     tp= t[old];          //先把原对应结点复制过来，再改关于x的新值
     if(l==r){
          tp.cnt += dx;
          return p;
     }
     int mid= l+r >>1;
     if(x<=mid)   //x在左子结点，右不动
          tp.lc= insert(tp.lc, l, mid, x, dx);
     else  //x在左子结点，右不动
          tp.rc= insert(tp.rc, mid+1, r, x, dx);
     tp.cnt= t[tp.lc].cnt + t[tp.rc].cnt;
     return p;
}
//关于[l,r]值域的根[p,q]的第k小值
int ask(int p, int q, int l, int r, int k){
     if(l==r)            //已经只有一种备选答案了
```

```
        return l;
    int mid= l+r >>1;
    ST &tp= t[p];
    ST &tq= t[q];
    int lcnt= t[tq.lc].cnt - t[tp.lc].cnt;
    if(k <= lcnt)//该去左边找
        return ask(tp.lc, tq.lc, l, mid, k);
    else  //该去右边找
        return ask(tp.rc, tq.rc, mid+1, r, k-lcnt);
}

int a0[MN],n;//数组a及其大小
int a[MN],newn,idx; //离散化后的a，临时坐标
int m,l,r,k; //查询次数和具体查询


for__(i,1,n)
        scanf("%d",a0+i),
        a[i] = a0[i];
    sort(a+1, a+1+n);
    newn= unique(a+1, a+1+n) -a-1;   //离散化，新n
for__(i,1,n)
        idx= lower_bound(a+1, a+1+newn, a0[i]) -a,    //离散化到idx
        rt[i]= insert(rt[i-1], 1, newn, idx, 1);      //在上一次的root
        上加1
    while(m--)
        scanf("%d%d%d",&l,&r,&k),
        printf("%d\n",a[ask(rt[l-1], rt[r], 1, newn, k)]);
```

3. 区间最大值树，待考证

```
int tot,rt[MN];     //每次更新的新树根
struct ST{
    int lc,rc;   //左右子结点编号
    int M; //区间最大值
}t[MN<<8];

//新建一个关于区间[l,r]的结点，返回其结点号
int build(int l,int r){
    int p= ++tot;//当前结点号
    ST &tp= t[p];//为了好写和好看，建一个引用
    if(l==r){
        tp.d= a[l];
        return p;
    }
    int mid= l+r >>1;
    tp.lc= build(l, mid);
    tp.rc= build(mid+1, r);
    tp.M= max(t[tp.lc].M, t[tp.rc].M);
    return p;
}
//把关于[l,r]的原old结点的值[x]改为v，返回新结点号
int insert(int old,int l,int r,int x,int v){
    int p= ++tot;//当前结点号
    ST &tp= t[p];//为了好写和好看，建一个引用
    tp= t[old];         //先把原对应结点复制过来，再改关于x的新值
    if(l==r){
```

```
            tp.M= v;
            return p;
        }
        int mid= l+r >>1;
        if(x<=mid)   //x在左子结点，右不动
            tp.lc= insert(tp.lc, l, mid, x, v);
        else  //x在左子结点，右不动
            tp.rc= insert(tp.rc, mid+1, r, x, v);
        tp.M= max(t[tp.lc].M, t[tp.rc].M);
        return p;
    }
```